Abstract
(DE 4131380)

A method of adapting an object-oriented application distributed among several operating system processes involves a processing method for code substitution from application sources to be compiled according to a stub concept. A configuration method is used to distribute instances of the application as objects or stub objects for modules of compiled application sources to be bound together. A communications method which is provided for a running process is used to call up methods for objects of the application with the aid of a stub method.

ADVANTAGE - Eliminates need for programmer to modify application sources which are to be compiled.

(19) BUNDESREPUBLIK DEUTSCHLAND

DEUTSCHES PATENTAMT

(12) **Offenlegungsschrift**

(10) **DE 41 31 380 A 1**

(51) Int. Cl.[5]: G 06 F 9/45

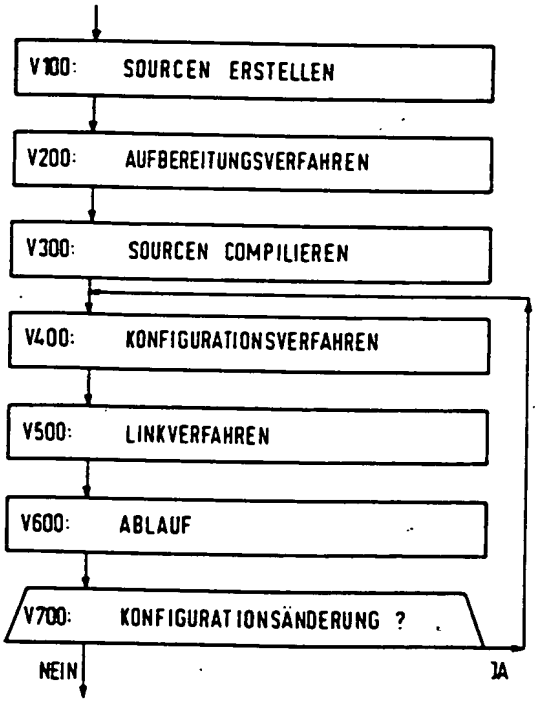
(21) Aktenzeichen: P 41 31 380.1
(22) Anmeldetag: 20. 9. 91
(43) Offenlegungstag: 25. 3. 93

(71) Anmelder:
Siemens AG, 8000 München, DE

(72) Erfinder:
Meyer, Walter, Dipl.-Ing.; Rothe, Oliver, Dipl.-Math., 8000 München, DE; Kneißl, Franz, Dr., 8500 Nürnberg, DE; Hubmann, Hans, Dipl.-Ing., 8524 Neunkirchen, DE; Beß, Rüdiger, 8510 Fürth, DE

(54) Verfahren zur Adaption einer objektorientierten Applikation

(57) Es wird bei einer objektorientierten Applikation beim Compilieren der Sourcen der Applikation ein Aufbereitungsverfahren, beim Binden ein Konfigurationsverfahren, sowie beim Ablauf ein Kommunikationsverfahren angewendet zum Aufrufen von Methoden für Objekte. Bei einer Änderung einer Systemkonfiguration ist eine Adaption der Sourcen nicht erforderlich. Dies gilt auch bei einer Erweiterung der objektorientierten Applikation.



DE 41 31 380 A 1

DE 41 31 380 A 1

Beschreibung

Die Erfindung betrifft ein Verfahren zur Adaption einer objektorientierten Applikation, so daß diese auf mehrere Betriebssystemprozesse verteilbar ist.

Objektorientierte Applikationen sind mittels einer objektorientierten Programmierung realisierbar. Ein objektorientiertes System besteht nicht nur aus Funktionen oder Prozeduren, welche einander aufrufen, etwa bei einer Programmierung mit den Programmiersprachen Fortran, Pascal, C, sondern es sind auch Objekte vorgesehen, die durch den Austausch von Nachrichten miteinander kommunizieren. Ein Beispiel für eine objektorientierte Programmiersprache ist die Programmiersprache C++, welche durch eine Erweiterung der weit verbreiteten Programmiersprache C entstanden ist. In der Programmiersprache C++ werden Klassen von Objekten durch Typdefinitionen vereinbart. Die Objekte gelten als Variablen eines Klassentyps und werden als Instanzen der Klasse bezeichnet. Jedes Objekt hat Instanzvariable als einen Satz von Daten. Auf die Instanzvariablen kann nur mittels bestimmter Funktionen zugegriffen werden, die in der jeweiligen Klassendefinition festgelegt werden. Diese der Klasse zugeordneten Zugriffsfunktionen heißen Methoden der Klasse. In der Programmiersprache C++ ist das Senden einer Nachricht an ein Objekt gleichbedeutend mit dem Aufruf einer Methode für dieses Objekt. Die Programmiersprache C++ unterstützt die Vererbung von Klassen. Die Instanzvariablen und die Methoden einer Klasse können von einer abgeleiteten Klasse durch Vererbung übernommen werden. Es können Methoden der Basisklasse mit dem Schlüsselwort virtual deklariert werden, so daß diese Methoden in einer abgeleiteten Klasse redefinierbar sind. Objekte sind durch Zeiger referenzierbar sowie dynamisch erzeugbar, so daß erst zur Laufzeit entscheidbar ist, welche Implementierung einer Methode tatsächlich ausgeführt wird. In der Programmiersprache C++ sind Applikationen auf genau einen Prozeß im Sinne des Betriebssystems begrenzt. Bei einer realen Anwendung kann es zu Problemen führen, wenn die Applikation eine bestimmte Größe überschreitet, da Betriebssystemprozesse nur eine begrenzte Größe annehmen können. Somit ist ohne zusätzliche Maßnahmen auch die Größe einer C++ Applikation begrenzt. Aus einigen Anwendungsbereichen, insbesondere der Automatisierungstechnik sowie der Telekommunikation, bestehen Anforderungen, welche eine Aufteilung der Applikation auf mehrere unabhängige, getrennt ausführbare und ladbare Betriebssystemprozesse verlangen. Eine derartige Aufteilung auf mehrere Betriebssystemprozesse, also keine light-weight Prozesse, ist mit den Sprachmitteln von C++ alleine nicht formulierbar. Eine derartige Aufteilung einer Applikation auf mehrere Betriebssystemprozesse bedingt eine wesentliche Erweiterung der Funktionalität, indem Mechanismen des Betriebssystems genutzt werden sollen, die die Kommunikation zwischen den Betriebssystemprozessen herstellen. Es ist denkbar, daß explizit vom Programmierer in die Applikation Aufrufe für eine Interprozeßkommunikation (IPC) eingefügt werden, sowie daß eigene IPC-Klassen verwendet werden. Der Softwareersteller hat in diesem Fall die Programmierung der IPC-Mechanismen selbst vorzunehmen. Die Aufteilung der Applikation auf die einzelnen Betriebssystemprozesse ist fest im Sourceprogramm eincodiert. Bei einer erforderlichen Änderung der Prozeßaufteilung sind vom Programmierer die Sourceprogramme zu modifizieren und erneut zu compilieren.

Es ist Aufgabe der Erfindung, ein Verfahren anzugeben, zur Adaption einer objektorientierten Applikation, welche auf mehrere Betriebssystemprozesse verteilbar ist, so daß insbesondere der Programmierer die zu compilierenden Sourcen der Applikation nicht selbst modifizieren muß.

Diese Aufgabe ist gelöst bei einem Verfahren zur Adaption einer objektorientierten Applikation, welche auf mehrere Betriebssystemprozesse verteilbar ist,

a) mit einem Aufbereitungsverfahren zur Codesubstitution von zu compilierenden Sourcen der Applikation gemäß einem Stub-Konzept,

b) mit einem Konfigurationsverfahren zur Verteilung von Instanzen der Applikation als Objekte oder Stub-Objekte für zu bindende Module aus compilierten Sourcen der Applikation,

c) mit einem beim Ablauf vorgesehenen Kommunikationsverfahren zum Aufrufen von Methoden für Objekte der Applikation mit Hilfe einer Stub-Methode.

Ausführbar ist ein bevorzugtes Verfahren mit zumindest einem von folgenden Verfahrensschritten des Aufbereitungsverfahrens:

d) es wird eine Klassendeklaration der Sourcen analysiert für eine Vergabe von Methoden-Identifikationen, so daß Methoden von verwendeten Klassen der Applikation eindeutig mittels der Methoden-Identifikation identifizierbar sind,

e) es werden die Klassen der Applikation mit einer generischen Methode ergänzt, mittels welcher jene Methode beim Ablauf lokal aufrufbar ist, welche durch einen Parameter der generischen Methode in Form der Methoden-Identifikation identifiziert wird,

f) es werden Basisklassen der Applikation um eine redifinierbar deklarierte Stub-Methode ergänzt,

g) es wird jeder von bei einem Ablauf vorgesehenen Aufrufen von einer Methode für ein Objekt der Klassen ersetzt durch einen beim Ablauf vorgesehenen Aufruf der Stub-Methode für das Objekt,

– so daß bei einem positiven Ergebnis des Aufrufs der Stub-Methode für das Objekt beim Ablauf ein Aufruf vorgesehen ist zu einem prozeßübergreifenden Versenden von einer Nachricht, durch welche in einem Remote-Prozeß der Aufruf der Methode des Objektes veranlaßt wird, indem die Methoden-Identifikation in der Nachricht enthalten ist,

– sowie daß bei einem negativen Ergebnis des Aufrufs der Stub-Methode für das Objekt beim Ablauf im Lokal-Prozeß der Aufruf der Methode des Objektes erfolgt,

h) es werden Hilfsdefinitionen für Methoden generiert, so daß zu jeder Methode der Applikation zumindest,

- ihr Klassenname,
- ihr Methodenname,
- ihre Parametertypen,
- ihr Parameterstring für ein Einpacken sowie Auspacken von Parametern, sowie,
- ihre Methoden-Identifikation definiert sind.

Ausführbar ist ein weiteres bevorzugtes Verfahren mit zumindest einem von folgenden Verfahrensschritten des Konfigurationsverfahrens:

k) es wird eine bestimmte Systemkonfiguration der Applikation analysiert für eine Verteilung der Objekte als die Instanzen der Applikation auf die Betriebssystemprozesse der Applikation,
m) es wird eine generische Instanzierungsfunktion generiert, mittels welcher beim Ablauf ein neues Objekt der Applikation instanzierbar ist,
- mit einer lokalen Instanzierung im Lokal-Prozeß bei einem gemäß der Systemkonfiguration lokal zu instanzierenden Objekt, so daß das negative Ergebnis beim Aufruf der Stub-Methode im Lokal-Prozeß für dieses lokal-instanzierte Objekt vorgesehen ist,
- sowie mit einer remoten Instanzierung bei einem gemäß der Systemkonfiguration remote zu instanzierenden Objekt, indem im Lokal-Prozeß ein prozeßübergreifender Anstoß vorgesehen ist zur Instanzierung dieses remote zu instanzierenden Objektes im Remote-Prozeß, sowie mit einer lokalen Instanzierung für ein zu diesem Objekt vorgesehenes lokales Stub-Objekt, dessen Stub-Methode im Lokal-Prozeß redefiniert wird, so daß das positive Ergebnis beim Aufruf der Stub-Methode für dieses als Stub-Objekt lokal instanzierte Objekt im Lokal-Prozeß vorgesehen ist,
n) es wird eine generische Löschfunktion generiert, mittels welcher beim Ablauf die Instanzierung von einem der Objekte der Applikation löschbar ist,
- mit einem lokalen Löschen im Lokal-Prozeß bei einem gemäß der Systemkonfiguration lokal-instanzierten Objekt,
- sowie mit einem remoten Löschen bei einem gemäß der Systemkonfiguration remote instanzierten Objekt, indem im Lokal-Prozeß ein prozeßübergreifender Anstoß vorgesehen ist zum Löschen dieses remote-instanzierten Objektes im Remote-Prozeß, sowie ein lokales Löschen von dem zu diesem Objekt vorgesehenen lokalen Stub-Objekt,
p) es werden Runfiles erzeugt, indem zu den Modulen bei jeder ladbaren Einheit Module,
- für die Instanzierungsfunktion,
- für die Löschfunktion sowie
- für die Hilfsdefinitionen für die Methoden der Applikation dazugebunden werden.

Ausführbar ist ein weiteres bevorzugtes Verfahren mit zumindest einem von folgenden Verfahrensschritten des Kommunikationsverfahrens:

r) es wird die Stub-Methode für eines der Objekte lokal aufgerufen, sowie im Falle des negativen Ergebnisses erfolgt ein lokaler Methodenaufruf für das Objekt,
s) es wird im Falle des positiven Ergebnisses für den lokalen Aufruf der Stub-Methode für das Objekt, aus den gebundenen Hilfsdefinitionen,
- die Methoden-Identifikation,
- das remote-instanzierte Objekt sowie
- Methodenparameter ermittelt,
t) es wird anhand des Parameterstrings eine Nachricht verpackt im Lokal-Prozeß,
u) es wird die Nachricht im Remote-Prozeß empfangen,
v) es wird im Remote-Prozeß nach dem Auspacken der Parameter das lokal instanzierte Objekt ermittelt,
w) es wird mittels der generischen Methode anhand der Methoden-Identifikation der Aufruf der dadurch identifizierten Methode ausgeführt.

Der Erfindung liegt die Idee zugrunde, daß die zu compilierenden Sourcen der Applikation in einem Aufbereitungsverfahren insbesondere mittels eines Präprozessors modifizierbar sind, so daß durch Codesubstitution Methodenaufrufe ersetzbar sind durch Codesequenzen, mittels derer beim Ablauf entscheidbar ist, ob ein prozeßübergreifendes Versenden einer Nachricht erforderlich ist, oder ob ein lokaler Aufruf erfolgen soll. Dies ist realisierbar, indem Hilfsdefinitionen für die Methoden generiert werden, so daß durch den Einsatz dieser speziellen Dateien eine Sicherung der Konsistenz des Systems bei der Applikation gewährleistet ist. Dies ist weiterhin erzielbar, indem ein Stub-Konzept angewendet wird, bei welchem Stub-Methoden und Stub-Objekte verwendet werden. Weiterhin ist dies erzielbar, indem eine Instanzierungsfunktion sowie eine Löschfunktion in einem Runtime-System eingesetzt wird. Dies ist weiterhin erzielbar, indem bei einem prozeßübergreifenden Kommunikationsverfahren beim Ablauf insbesondere eine generische Methode eingesetzt wird, von welcher anhand einer Methoden-Identifikation der dadurch identifizierte Methodenaufruf ausführbar ist.

In einer vorteilhaften Weise braucht der Programmierer die Sourcen nicht selbst modifizieren.

In einer vorteilhaften Weise kann der Programmierer die Sourcen der Applikation beispielsweise mit den Sprachmitteln der Programmiersprache C++ erstellen.

In einer vorteilhaften Weise ist mittels der Codesubstitution beim Aufbereitungsverfahren nur eine einmalige Modifikation der Sourcen mit deren anschließender Compilierung ausreichend. Bei einer Änderung der Systemkonfiguration, also bei einer Änderung der Verteilung der Applikation auf mehrere Betriebssystemprozesse, ist

eine Änderung der Sourcen der Applikation nicht erforderlich, so daß auch eine neue Compilierung nicht erforderlich ist.

In einer vorteilhaften Weise wird mittels der Hilfsdefinitionen für eine bestimmte Systemkonfiguration die Konsistenz des Gesamtsystems der objektorientierten Applikation sichergestellt.

Die Erfindung wird anhand der Figuren, in welchen Ausführungsbeispiele enthalten sind, näher erläutert.

Die **Fig.** 1 zeigt ein erfindungsgemäßes Verfahren zur Adaption einer objektorientierten Applikation, welche auf mehrere Betriebssystemprozesse verteilbar ist.

Die **Fig.** 2 zeigt ein anderes Verfahren zur Adaption einer objektorientierten Applikation, welche auf mehrere Betriebsprozesse verteilbar ist.

Die **Fig.** 3 zeigt ein Aufbereitungsverfahren für das erfindungsgemäße Verfahren zur Adaption einer objektorientierten Applikation, welche auf mehrere Betriebssystemprozesse verteilbar ist.

Die **Fig.** 4 zeigt ein Konfigurationsverfahren für das erfindungsgemäße Verfahren zur Adaption einer objektorientierten Applikation, welche auf mehrere Betriebssystemprozesse verteilbar ist.

Die **Fig.** 5 zeigt ein prozeßübergreifendes Kommunikationsverfahren für das erfindungsgemäße Verfahren zur Adaption einer objektorientierten Applikation, welche auf mehrere Betriebssystemprozesse verteilbar ist.

Die **Fig.** 6 zeigt einen Methodenaufruf.

Die **Fig.** 7 zeigt eine Codesequenz nach einer Codesubstitution des Methodenaufrufs.

Die **Fig.** 8 zeigt einen prozeßübergreifenden Sendeaufruf für eine Nachricht als einen Teil der Codesequenz.

Die **Fig.** 9 zeigt einen Aufruf einer Stub-Methode als ein Teil der Codesequenz.

Wie die **Fig.** 1 zeigt, besteht ein Ausführungsbeispiel für das erfindungsgemäße Verfahren zur Adaption einer objektorientierten Applikation, welche auf mehrere Betriebssystemprozesse verteilbar ist, aus den Verfahrensschritten V100, V200, V300, V400, V500, V600, sowie V700.

Es wird der Verfahrensschritt V100 ausgeführt. Vom Programmierer werden die zu compilierenden Sourcen der Applikation erstellt.

Es wird der Verfahrensschritt V200 ausgeführt. Für die zu compilierenden erstellten Sourcen der Applikation wird ein Aufbereitungsverfahren durchgeführt. Dabei kann ein Präprozessor vorgesehen sein zur Durchführung des Aufbereitungsverfahrens. Es werden Klassendeklarationen der Sourcen der Applikation analysiert. Es werden Methoden-Identifikationen aufbereitet. Die Klassen der Applikation werden um die generische Methode ergänzt. Es werden die Methodenaufrufe in den Sourcen der Applikation modifiziert, indem mittels Codesubstution Codesequenzen eingesetzt werden anstelle der Methodenaufrufe. Es werden Hilfsdefinitionen für die Methoden der Applikation generiert.

Es wird der Verfahrensschritt V300 ausgeführt. Die vom Aufbereitungsverfahren modifizierten Sourcen werden compiliert.

Es folgt der Verfahrensschritt V400. Es wird ein Konfigurationsverfahren ausgeführt. Es wird eine Konfigurationsdatei analysiert. Es wird eine Instanzierungsfunktion generiert. Es wird eine Löschfunktion generiert. Es werden Hilfsdefinitionen für die Methoden der Applikation mitgebunden. Es werden Runfiles erzeugt.

Es folgt der Verfahrensschritt V500. Es wird ein Linkverfahren ausgeführt. Es werden ablauffähige Phasen der Betriebssystemprozesse gebunden.

Es folgt der Verfahrensschritt V600. Es wird ein Ablauf der Betriebssystemprozesse der Applikation ausgeführt.

Es folgt der Verfahrensschritt V700. Es wird untersucht, ob eine Konfigurationsänderung erforderlich ist. Falls dies der Fall ist, folgt der Verfahrensschritt V400. Gemäß der geänderten Systemkonfiguration wird das Konfigurationsverfahren erneut durchgeführt. Danach folgt der Verfahrensschritt V500, bei welchem das Linkverfahren erneut durchgeführt wird. Danach folgt der Verfahrensschritt V600, bei welchem der Ablauf der Applikation gemäß der geänderten Systemkonfiguration erfolgt.

Wie die **Fig.** 2 zeigt, besteht ein Ausführungsbeispiel für ein anderes Verfahren zur Adaption einer objektorientierten Applikation aus den Verfahrensschritten V101, V301, V501, V601, V701 sowie V801.

Es wird der Verfahrensschritt V101 ausgeführt. Von einem Programmierer werden die Sourcen der Applikation erstellt. Je nach Aufteilung der Applikation auf mehrere Betriebssystemprozesse werden vom Programmierer explizit die Aufrufe für eine prozeßübergreifende Kommunikation in den Sourcen der Applikation eingetragen.

Es folgt der Verfahrensschritt V301. Die vom Programmierer erstellten Sourcen werden compiliert.

Es folgt der Verfahrensschritt V501. Für die compilierten Sourcen der Applikation wird das Linkverfahren durchgeführt.

Es folgt der Verfahrensschritt V601. Es erfolgt der Ablauf der Applikation.

Es folgt der Verfahrensschritt V701. Es wird geprüft, ob eine Konfigurationsänderung erforderlich ist. Falls dies der Fall ist, folgt der Verfahrensschritt V801. Gemäß einer neuen Systemkonfiguration werden vom Programmierer die Sourcen der Applikation modifiziert, indem gemäß der neuen Verteilung der Applikation auf einzelne Betriebssystemprozesse in den Sourcen explizit Aufrufe für eine prozeßübergreifende Kommunikation eingefügt werden. Danach folgt erneut der Verfahrensschritt V301, bei welchem die soeben modifizierten Sourcen compiliert werden.

Danach folgt der Verfahrensschritt V501, bei welchem das Linkverfahren erneut durchgeführt wird.

Danach folgt der Verfahrensschritt V601, bei welchem der Ablauf der Applikation gemäß der neuen Systemkonfiguration erfolgt.

Wie die **Fig.** 3 zeigt, besteht ein Ausführungsbeispiel für das Aufbereitungsverfahren des Verfahrensschrittes V200 aus den Verfahrensschritten V210, V220, V230, V235, V240 sowie V250.

Es wird der Verfahrensschritt V210 ausgeführt. Es wird eine Klassendeklaration der Sourcen der Applikation analysiert für eine Vergabe von Methoden-Identifikationen, so daß die Methoden von den verwendeten Klassen

der Applikation systemweit eindeutig mittels der Methoden-Identifikation identifizierbar sind.

Es folgt der Verfahrensschritt V220. Es wird eine Protokoll-Informations-Datei aufbereitet, in welcher Hilfsdefinitionen für die Methoden der Applikation enthalten sind, so daß zu jeder Methode der Applikation zumindest ihr Klassenname, ihr Methodenname, ihre Parametertypen, ihr Parameterstring für ein Einpacken sowie Auspacken von Parametern sowie ihre Methoden-Identifikation definiert sind.

Es folgt der Verfahrensschritt V230. Es werden die Klassen der Applikation um die generische Methode ergänzt. Mittels der generischen Methode ist anhand der Methoden-Identifikation ein Aufruf der dadurch identifizierten Methode ausführbar.

Es folgt der Verfahrensschritt V235. Es werden die Basisklassen der Applikation um eine redefinierbar deklarierte Stub-Methode ergänzt.

Es folgt der Verfahrensschritt V240. Es werden in den Sourcen die Methodenaufrufe modifiziert mittels einer Codesubstitution.

Es folgt der Verfahrensschritt V250. In einer Hilfsdefinitionsdatei werden die Hilfsdefinitionen für die Methoden generiert und aufbereitet.

Wie die Fig. 4 zeigt, besteht ein Ausführungsbeispiel für das Konfigurationsverfahren des Verfahrensschrittes V400 aus den Verfahrensschritten V410, V420, V430, V440 sowie V450.

Es wird der Verfahrensschritt V410 ausgeführt. Es wird eine bestimmte Systemkonfiguration der Applikation anhand einer Konfigurationsdatei analysiert, mittels welcher eine Verteilung der Objekte der Applikation auf die Betriebssystemprozesse der Applikation vorgenommen wird.

Es folgt der Verfahrensschritt V420. Es wird eine generische Instanzierungsfunktion generiert, mittels welcher beim Ablauf die Objekte der Applikation instanzierbar sind.

Es folgt der Verfahrensschritt V430. Es wird eine generische Löschfunktion generiert, mittels welcher beim Ablauf die Instanzierung der Objekte der Applikation löschbar ist.

Es folgt der Verfahrensschritt V440. Es werden Module für die Hilfsdefinitionen für die Methoden der Applikation gebunden.

Es folgt der Verfahrensschritt V450. Es werden Runfiles erzeugt, indem zu den Modulen bei jeder ladbaren Einheit Module für die Instanzierungsfunktion, für die Löschfunktion sowie für die Hilfsdefinitionen für die Methoden der Applikation dazugebunden werden.

Wie die Fig. 5 zeigt, besteht als ein Teil des Verfahrensschrittes V600 ein Ausführungsbeispiel von einem Verfahrensschritt V605 für einen Ablauf zum Aufrufen einer Methode für ein Objekt der Applikation aus den Verfahrensschritten V610 bis V680.

In einem Lokal-Prozeß wird der Verfahrensschritt V610 ausgeführt. Es wird ein lokaler Adressat ermittelt. Dieser ist im Falle eines lokalen Objektes die Instanz selbst. Im Falle eines remoten Objektes ist der lokale Adressat das zugehörige Stub-Objekt.

Es folgt der Verfahrensschritt V620. Es wird für den lokalen Adressaten die Stub-Methode aufgerufen.

Es folgt der Verfahrensschritt V621. Es wird überprüft, ob das Ergebnis beim Aufruf der Stub-Methode für den lokalen Adressaten positiv ist.

Falls das Ergebnis negativ ist, folgt der Verfahrensschritt V680, und es erfolgt der Aufruf der Methode für den lokalen Adressaten, welcher in diesem Fall das instanzierte lokale Objekt ist.

Bei einem positiven Ergebnis folgt der Verfahrensschritt V640. Für die aufzurufende Methode wird deren Methoden-Identifikation ermittelt.

Es folgt der Verfahrensschritt V641. Es wird der remote Adressat ermittelt, welcher in diesem Fall das remote instanzierte Objekt ist.

Es folgt der Verfahrensschritt V642. Es werden die Parameter für den remoten Aufruf der Methode verpackt.

Es folgt der Verfahrensschritt V670. Es erfolgt eine Interprozeßkommunikation (IPC) zwischen dem Lokal-Prozeß und dem Remote-Prozeß.

Im Remote-Prozeß folgt der Verfahrensschritt V650. Es werden die Parameter für den Aufruf der Methode ausgepackt.

Es folgt der Verfahrensschritt V651. Es wird der lokale Adressat im Remote-Prozeß ermittelt. Dieser ist in diesem Fall das im Remote-Prozeß instanzierte Objekt.

Es folgt der Verfahrensschritt V652. Es erfolgt ein Aufruf der generischen Methode, bei welcher mittels der Methoden-Identifikation die aufzurufende Methode ermittelt wird.

Es folgt der Verfahrensschritt V660. Es erfolgt der Aufruf der Methode für das im Remote-Prozeß instanzierte Objekt.

Mittels der Interprozeßkommunikation des Verfahrensschrittes V670 wird nach dem Remote-Prozeß der Lokal-Prozeß fortgesetzt.

Wie die Fig. 6 zeigt, besteht ein Ausführungsbeispiel für einen lokalen Aufruf einer Methode für ein Objekt aus einem Objektpointer, einem Methodennamen sowie aus Parametern. Der Objektpointer bildet dabei eine Referenz auf das Objekt, an welches eine Nachricht gesendet wird. Der Methodenname bildet dabei eine Bezeichnung für die Nachricht. Die Parameter bilden dabei einen Parameterteil für die Nachricht.

Gemäß vereinbarter Notation gilt beispielsweise folgende Schreibweise:

<Objektpointer> → <Methodenname> (<Parameter>)

<Objektpointer>:	Referenz auf das Objekt, an das die Nachricht gesendet wird
<Methodenname>:	Bezeichnung der Nachricht
<Parameter>:	Parameterteil für die Nachricht

Beispielsweise bei der Programmiersprache C++ ist deren Nachrichtenmechanismus so ausgelegt, daß die Adressierung von Objekten innerhalb eines Betriebssystemprozesses unterstützt wird. Applikationen, welche auf mehrere unabhängige Betriebssystemprozesse verteilt werden sollen, benötigen daher einen erweiterten Nachrichtenmechanismus. Diese Erweiterung des Nachrichtenmechanismus soll nicht durch eine Erweiterung der Programmiersprache erfolgen, da auf diese Weise eine Portabilität eingeschränkt ist. Diese Erweiterung des Nachrichtenmechanismus soll in einer solchen Weise vorgenommen werden, daß eine Portabilität gewährleistet ist. Zusätzlich zu dem in der Programmiersprache verankerten prozeßlokalen Nachrichtenmechanismus soll noch ein Mechanismus für einen prozeßübergreifenden Nachrichtenaustausch in Form einer Interprozeßkommunikation (IPC) bereitgestellt werden. Dabei sollen keine neuen Sprachmittel benötigt werden, so daß eine Compilererweiterung nicht erforderlich ist. Von einem Präprozessor soll in einem Aufbereitungsverfahren für die zu compilierenden Sourcen der Applikation ein derartiger Methodenaufruf mittels einer Codesubstitution ersetzt werden durch eine Codesequenz, in welcher zusätzlich implizite Aufrufe für den prozeßübergreifenden Nachrichtenaustausch (IPC) enthalten sind. Mittels dieser Codesequenz soll sowohl der prozeßlokale Nachrichtenaustausch als auch ein erweiterter Nachrichtenaustausch über Prozeßgrenzen hinweg erlaubt werden.

Wie die **Fig. 7** zeigt, besteht ein Ausführungsbeispiel für eine derartige Codesequenz aus einem lokalen Aufruf einer Stub-Methode Vstub, so daß bei einem positiven Ergebnis dieses Aufrufs eine für die Interprozeßkommunikation (IPC) erweiterte Form des Aufrufs vorgesehen ist, sowie daß bei einem negativen Ergebnis ein lokaler Methodenaufruf vorgesehen ist, welcher gleich ist dem in der **Fig. 6** dargestellten Methodenaufruf, welcher durch die in der **Fig. 7** dargestellte Codesequenz bei der Codesubstitution ersetzt wird.

Gemäß vereinbarter Notation gilt beispielsweise folgende Schreibweise:

((<Objektpointer> → Vstub())?(SX_SEND((_CSXobj*)
<Objektpointer>, <Methoden-ID>, <Parameterstring>, <Parameter>):
(<Objektpointer> → <Methodenname> (<Parameter>))

<Methoden-ID>: Methoden-Identifikation zur systemweit eindeutigen Identifizierung von Methoden von Klassen der Applikation

<Parameterstring>: Zeichenstring zur Identifizierung von Parametern beim Einpacken sowie Auspacken von Parametern

Dabei dient die Methoden-Identifikation zur systemweit eindeutigen Identifizierung von Methoden für Klassen der Applikation. Es ist ein Parameterstring vorgesehen, welcher in Form eines Zeichenstrings zur Identifizierung von Parametern beim Einpacken sowie Auspacken von Parametern dient.

Wie die **Fig. 8** zeigt, enthält ein Ausführungsbeispiel einer derart erweiterten Form des Aufrufs für die Interprozeßkommunikation (IPC) einen Aufruf zum prozeßübergreifenden Versenden der Nachricht, einen Objektpointer, eine MethodenIdentifikation, einen Parameterstring sowie Parameter.

Gemäß vereinbarter Notation gilt beispielsweise folgende Schreibweise:

(SX_SEND((_CSXobj*) <Objektpointer>, <Methoden-ID>, <Parameterstring>, <Parameter>)

Ob dieser Aufruf tatsächlich aktiviert wird, wird zur Laufzeit entschieden durch eine Auswertung des Ergebnisses, welches als eine Abfrage mittels des Aufrufs der Stub-Methode ermittelt wird.

Wie die **Fig. 9** zeigt, besteht ein Ausführungsbeispiel für einen Aufruf der Stub-Methode aus einem Objektpointer sowie aus dem Methodennamen Vstub.

Gemäß vereinbarter Notation gilt beispielsweise folgende Schreibweise:

(<Objektpointer> → Vstub())

Vor dem Versenden der Nachricht gemäß des Methodennamens an das referenzierte Objekt wird jedesmal ermittelt, ob ein prozeßlokaler Methodenaufruf stattfinden soll, oder ob ein mittels der Interprozeßkommunikation (IPC) erweiterter Aufruf durchgeführt wird. Im Falle eines IPC-Aufrufes kommt die Funktion SX-SEND zum Tragen, welche mit den geeigneten Parametern zu versorgen ist. Diese Parameterversorgung soll bei der Codesubstitution automatisiert vom Präprozessor vorgenommen werden.

In einer vorteilhaften Weise erfolgt die Codesubstitution nach einfachen Regeln, so daß diese automatisierbar ist, und beispielsweise von einem Tool durchgeführt werden kann.

Zusammen mit der Codesubstitution für die Nachrichtenaufrufe sollen auch noch diejenigen Codeteile substituiert werden, welche die Instanzierung von Klassen sowie das Entfernen von Objekten aus dem System betreffen.

Die Entscheidung, welche Form des Nachrichtenmechanismus jeweils zum Tragen kommt, also ein prozeßlokales Versenden der Nachricht oder ein prozeßübergreifendes Versenden der Nachricht, soll zur Laufzeit des Programmes getroffen werden, und zwar in Abhängigkeit von einer aktuellen Systemkonstellation, welche bei objektorientierten Softwaresystemen dynamisch veränderbar ist. Um eine hohe Performanz zu gewährleisten, soll diese Abfrage sehr schnell vonstatten gehen und deshalb prozeßlokal erfolgen. In jedem Betriebssystemprozeß soll jeweils eine Instanz existieren, welche darüber Auskunft geben kann, ob sich ein adressiertes Objekt gerade im jeweiligen Betriebssystemprozeß des Nachrichtensenders befindet, oder falls es in einem anderen Betriebssystemprozeß existiert, eine Wegeinformation bereitstellen zur Ermöglichung der Adressierung. Dies soll mittels eines Stub-Konzeptes erfolgen.

Bei einem derartigen Stub-Konzept sollen in einer verteilten Applikation in denjenigen Betriebssystemprozessen, welche nicht selbst ein reales Objekt enthalten, Stellvertreter-Objekte als Stub-Objekte anstelle der realen Objekte vorhanden sein. Diese Stub-Objekte stehen dann im Falle einer Anforderung an das reale Objekt als Ansprechpartner lokal zur Verfügung und können entweder dessen Aufgaben direkt übernehmen oder die geforderte Dienstleistung an das reale Objekt weitervermitteln. Diese Stub-Objekte sollen sich gegenüber den anderen Objekten exakt wie die realen Objekte verhalten.

Eine derartige Aufgabe oder eine mögliche Dienstleistung ist beispielsweise die Auskunft darüber, ob es sich bei einem bestimmten Objekt um die Instanz selbst handelt oder um das Stub-Objekt. Falls es sich dabei um die Instanz selbst handelt, kann die Nachricht in der beispielsweise bei der Programmiersprache C++ üblichen Form direkt an das adressierte Objekt gesendet werden. Anderenfalls soll das Stub-Objekt alle notwendigen Informationen darüber bereitstellen, um die entsprechende Anforderung an das reale Objekt weiterzuleiten. Im Bedarfsfall kann jedes Objekt, einschließlich der Stub-Objekte, darüber Auskunft geben, ob es selbst ein Stub-Objekt ist oder nicht. Zu diesem Zweck wird eine Methode, die Stub-Methode Vstub(), als virtuelle Methode in der Basisklasse aller Applikationsklassen bereitgestellt. Durch den Vererbungsmechanismus erhalten alle Objekte der Applikation automatisch die Fähigkeit, Auskunft darüber zu geben, ob sie Stub-Objekte sind oder reale Objekte. Es wird in der Basisklasse aller Applikationsklassen die Stub-Methode bereitgestellt, welche im Falle einer Aktivierung ein negatives Ergebnis liefert. Die Stub-Methode ist redefinierbar zu deklarieren, indem die Stub-Methode beispielsweise bei der Programmiersprache C++ das Schlüsselwort virtual enthält. Für Stub-Klassen wird die Stub-Methode redefiniert, so daß sie ein positives Ergebnis liefert. Demnach liefern Stub-Objekte auf die Anfrage Vstub() gemäß der Stub-Methode ein anderes Ergebnis als die Nicht-Stub-Objekte, bei welchen eine Default-Implementierung zum Tragen kommt.

Eine Systemkonfiguration, also eine Aufteilung der Objekte auf Betriebssystemprozesse, braucht erst nach dem Übersetzen der Quellprogramme festgelegt werden. Erst zur Laufzeit soll entschieden werden, welcher Nachrichtenmechanismus jeweils zum Tragen kommt. Dies gilt auch für den Vorgang der Instanzierung von Klassen sowie das Entfernen von Objekten aus dem laufenden System. Dies kann nicht statisch vom Compiler erledigt werden, sondern soll vom Laufzeitsystem ausgeführt werden.

Ebenso wie bei der Codesubstitution für die Nachrichtenaufrufe werden auch die Aufrufe NEW und DELETE in der Source vom Präprozessor durch die Funktionen SX—NEWobj als Instanzierungsfunktion und SX—DELETE als Löschfunktion ersetzt. Diese Funktionen werden im Laufzeitsystem ausgeführt. Diese Funktionen werden zu jeder ladbaren Einheit dazugebunden. Sie inkorporieren die Informationen darüber, ob ein Objekt lokal oder remote indistanziert oder gelöscht werden soll. Bei der Instanzierung wird demzufolge entweder der Operator NEW beispielsweise bei der Programmiersprache C++ verwendet, oder es wird über die Interprozeßkommunikation die Instanzierung des Objektes in einem anderen Betriebssystemprozeß angestoßen und lokal wird dabei nur ein Stub-Objekt erzeugt. Die Löschfunktion entscheidet zur Laufzeit, ob das Objekt, auf das sie angewendet wird, ein Stub-Objekt oder ein reales Objekt ist. Abhängig davon wird entweder der Operator DELETE beispielsweise bei der Programmiersprache C++ verwendet oder es wird über die Interprozeßkommunikation das Löschen des Objektes in einem anderen Betriebssystemprozeß angestoßen und lokal wird das Stub-Objekt gelöscht. Die Implementierung der Instanzierungsfunktion und der Löschfunktion wird aufgrund von Angaben in einer Konfigurationsdatei generiert.

Beim prozeßübergreifenden Kommunikationsmechanismus gehen Methodenaufrufe, also auch Aufrufe von Konstrukturen und Destruktoren, über Prozeßgrenzen hinweg, so daß diese Aufrufe in versendbare Daten umzuwandeln sind. Dabei werden die Methoden der verwendeten Klassen durch Methoden-Identifikationen identifiziert. Diese Methoden-Identifikationen werden beispielsweise mittels einer Protokoll-Informations-Datei systemweit eindeutig vergeben und mittels der generischen Methode ausgewertet, welche jedes Objekt besitzt. Die Protokoll-Informations-Datei bildet eine systemweite Datenbasis über Methoden und ihre Methoden-Identifikationen. Diese wird bei jeder Codesubstitution ausgewertet und soweit erforderlich vervollständigt. Die Protokoll-Informations-Partei enthält für jede Methode der Applikation folgende Daten:

- Klassenname,
- Methodenname,
- Typen der Parameter der Methode (diese Angabe ist notwendig, da beispielsweise in der Programmiersprache C++ Methoden nicht allein aufgrund von Klassennamen und Methodennamen, sondern erst durch die Parametertypen eindeutig identifiziert werden können),
- Parameterstring (beispielsweise verwendet für das Einpacken sowie Auspacken der Parameter),
- Methoden-Identifikation.

Bei der Codesubstitution wird die Definition jeder verwendeten Klasse um die generische Methode erweitert. Diese bildet die Methoden-Identifikationen auf lokale Methodenaufrufe ab. Jeder Methoden-Aufruf aus einem anderen Prozeß führt im Empfängerprozeß zunächst zu einem Aufruf der generischen Methode mit der Methoden-Identifikation als Parameter.

Beim prozeßübergreifenden Versenden der Nachricht werden die Parameter des Methodenaufrufs in einer Datenstruktur verpackt und verschickt. Die Funktionen SX-SEND und die generische Methode sind für das Einpacken sowie Auspacken der Parameter vorgesehen. Dazu werden der Funktion SX-SEND Informationen über Methodenparameter codiert übergeben, beispielsweise in Form eines Strings. Dieser Informationsstring, also der Parameterstring, wird bei der Codesubstitution ebenfalls vom Präprozessor aufbereitet.

Somit wird bei einer objektorientierten Applikation beim Compilieren der Applikation ein Aufbereitungsverfahren, beim Binden ein Konfigurationsverfahren sowie beim Ablauf ein Kommunikationsverfahren angewendet zum Aufrufen von Methoden für Objekte. Bei einer Änderung einer Systemkonfiguration ist eine Adaption

der Sourcen nicht erforderlich. Dies gilt auch bei einer Erweiterung der objektorientierten Applikation.

Patentansprüche

1. Verfahren zur Adaption einer objektorientierten Applikation, welche auf mehrere Betriebssystemprozesse verteilbar ist,
   a) mit einem Aufbereitungsverfahren (V200) zur Codesubstitution von zu compilierenden Sourcen der Applikation, gemäß einem Stub-Konzept,
   b) mit einem Konfigurationsverfahren (V400) zur Verteilung von Instanzen der Applikation als Objekte oder Stub-Objekte für zu bindende Module aus compilierten Sourcen der Applikation,
   c) mit einem beim Ablauf vorgesehenen Kommunikationsverfahren (V605) zum Aufrufen von Methoden für Objekte der Applikation mit Hilfe einer Stub-Methode.

2. Verfahren nach Anspruch 1, mit zumindest einem von folgenden Verfahrensschritten des Aufbereitungsverfahrens (V200):
   d) es wird eine Klassendeklaration der Sourcen analysiert (V210) für eine Vergabe von Methoden-Identifikationen, so daß Methoden von verwendeten Klassen der Applikation eindeutig mittels der Methoden-Identifikation identifizierbar sind,
   e) es werden die Klassen der Applikation mit einer generischen Methode ergänzt (V230), mittels welcher jene Methode beim Ablauf lokal aufrufbar ist, welche durch einen Parameter der generischen Methode in Form der Methoden-Identifikation identifiziert wird,
   f) es werden Basisklassen der Applikation um eine redifinierbar deklarierte Stub-Methode ergänzt (V235),
   g) es wird jeder von bei einem Ablauf vorgesehenen Aufrufen von einer Methode für ein Objekt der Klassen ersetzt (V240) durch einen beim Ablauf vorgesehenen Aufruf der Stub-Methode für das Objekt,
   – so daß bei einem positiven Ergebnis des Aufrufs der Stub-Methode für das Objekt beim Ablauf ein Aufruf vorgesehen ist zu einem prozeßübergreifenden Versenden von einer Nachricht, durch welche in einem Remote-Prozeß der Aufruf der Methode des Objektes veranlaßt wird, indem die MethodenIdentifikation in der Nachricht enthalten ist,
   – sowie daß bei einem negativen Ergebnis des Aufrufs der Stub-Methode für das Objekt beim Ablauf im Lokal-Prozeß der Aufruf der Methode des Objektes erfolgt,
   h) es werden Hilfsdefinitionen für Methoden generiert (250), so daß zu jeder Methode der Applikation zumindest
   – ihr Klassenname,
   – ihr Methodenname,
   – ihre Parametertypen,
   – ihr Parameterstring für ein Einpacken sowie Auspacken von Parametern, sowie
   – ihre Methoden-Identifikation definiert sind.

3. Verfahren nach Anspruch 2 mit zumindest einem von folgenden Verfahrensschritten des Konfigurationsverfahrens (V400):
   k) es wird eine bestimmte Systemkonfiguration der Applikation analysiert (V410) zur Verteilung der Objekte als die Instanzen der Applikation auf die Betriebssystemprozesse der Applikation,
   m) es wird eine generische Instanzierungsfunktion generiert (V420), mittels welcher beim Ablauf ein neues Objekt der Applikation instanzierbar ist,
   – mit einer lokalen Instanzierung im Lokal-Prozeß bei einem gemäß der Systemkonfiguration lokal zu instanzierenden Objekt, so daß das negative Ergebnis beim Aufruf der Stub-Methode im Lokal-Prozeß für dieses lokal-instanzierte Objekt vorgesehen ist,
   – sowie mit einer remoten Instanzierung bei einem gemäß der Systemkonfiguration remote zu instanzierenden Objekt, indem im Lokal-Prozeß ein prozeßübergreifender Anstoß vorgesehen ist zur Instanzierung dieses remote zu instanzierenden Objektes im Remote-Prozeß, sowie mit einer lokalen Instanzierung für ein zu diesem Objekt vorgesehenes lokales Stub-Objekt, dessen Stub-Methode im Lokal-Prozeß redefiniert wird, so daß das positive Ergebnis beim Aufruf der Stub-Methode für dieses als Stub-Objekt lokal instanzierte Objekt im Lokal-Prozeß vorgesehen ist,
   n) es wird eine generische Löschfunktion generiert (V430), mittels welcher beim Ablauf die Instanzierung von einem der Objekte der Applikation löschbar ist,
   – mit einem lokalen Löschen im Lokal-Prozeß bei einem gemäß der Systemkonfiguration lokal-instanzierten Objekt,
   – sowie mit einem remoten Löschen bei einem gemäß der Systemkonfiguration remote instanzierten Objekt, indem im Lokal-Prozeß ein prozeßübergreifender Anstoß vorgesehen ist zum Löschen dieses remote-instanzierten Objektes im Remote-Prozeß, sowie ein lokales Löschen von dem zu diesem Objekt vorgesehenen lokalen Stub-Objekt,
   p) es werden Runfiles erzeugt (V450), indem zu den Modulen bei jeder ladbaren Einheit Module
   – für die Instanzierungsfunktion,
   – für die Löschfunktion sowie
   – für die Hilfsdefinitionen für die Methoden der Applikation dazugebunden werden.

4. Verfahren nach Anspruch 3 mit zumindest einem von folgenden Verfahrensschritten des Kommunikationsverfahrens (V605):
   r) es wird die Stub-Methode für eines der Objekte lokal aufgerufen (V620), sowie im Falle des negativen

Ergebnisses erfolgt ein lokaler Methodenaufruf für das Objekt (V680),
s) es wird im Falle des positiven Ergebnisses für den lokalen Aufruf der Stub-Methode für das Objekt, aus den gebundenen Hilfsdefinitionen
- die Methoden-Identifikation (V640),
- das remote-instanzierte Objekt (V641) sowie
- Methodenparameter ermittelt,

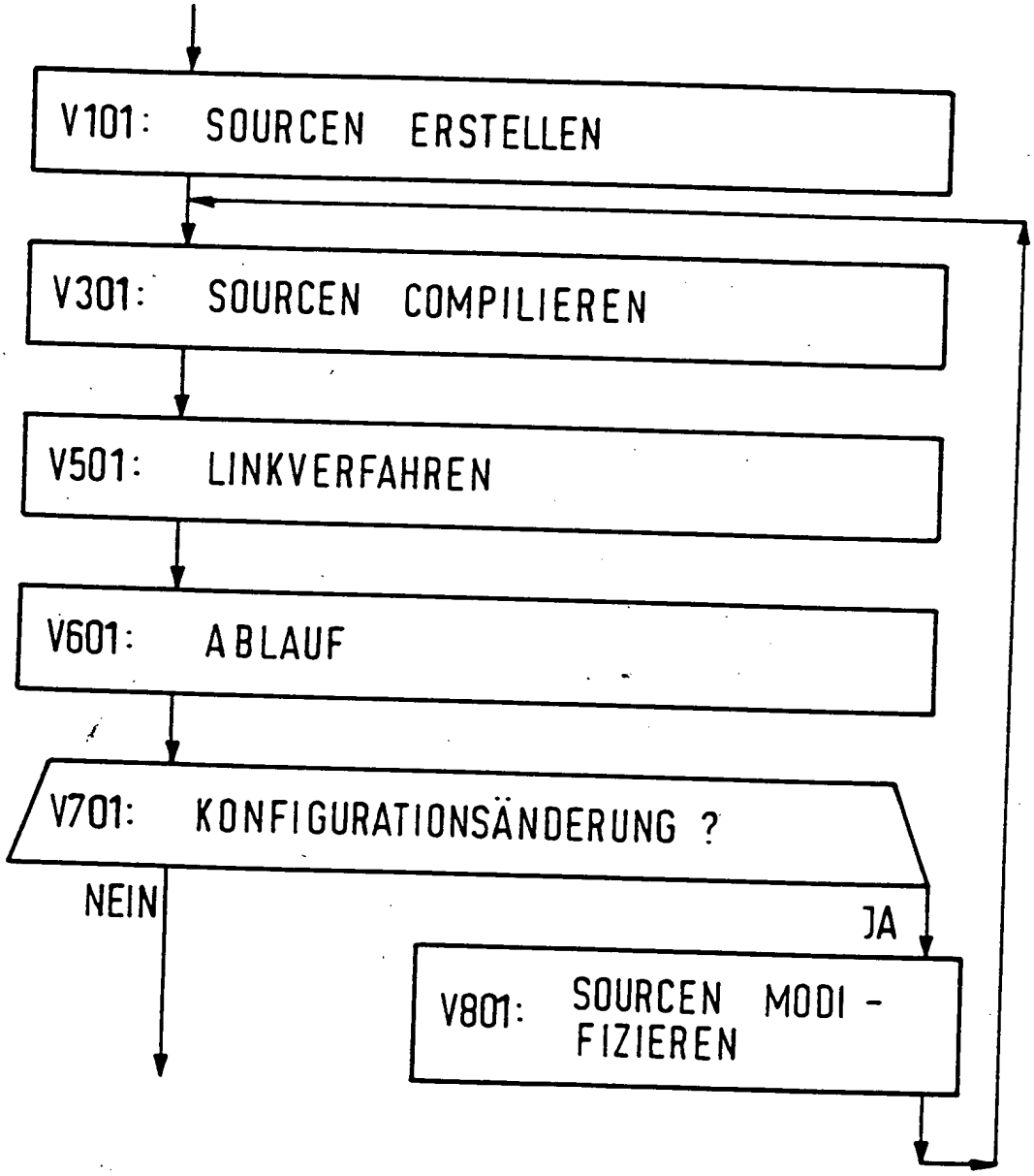
t) es wird anhand des Parameterstrings eine Nachricht verpackt im Lokal-Prozeß (V642),
u) es wird die Nachricht im Remote-Prozeß empfangen (V670),
v) es wird im Remote-Prozeß nach dem Auspacken der Parameter (V650) das lokal instanzierte Objekt ermittelt (V651),
w) es wird mittels der generischen Methode anhand der Methoden-Identifikation (V652) der Aufruf der dadurch identifizierten Methode ausgeführt (V660).

Hierzu 6 Seite(n) Zeichnungen

– Leerseite –

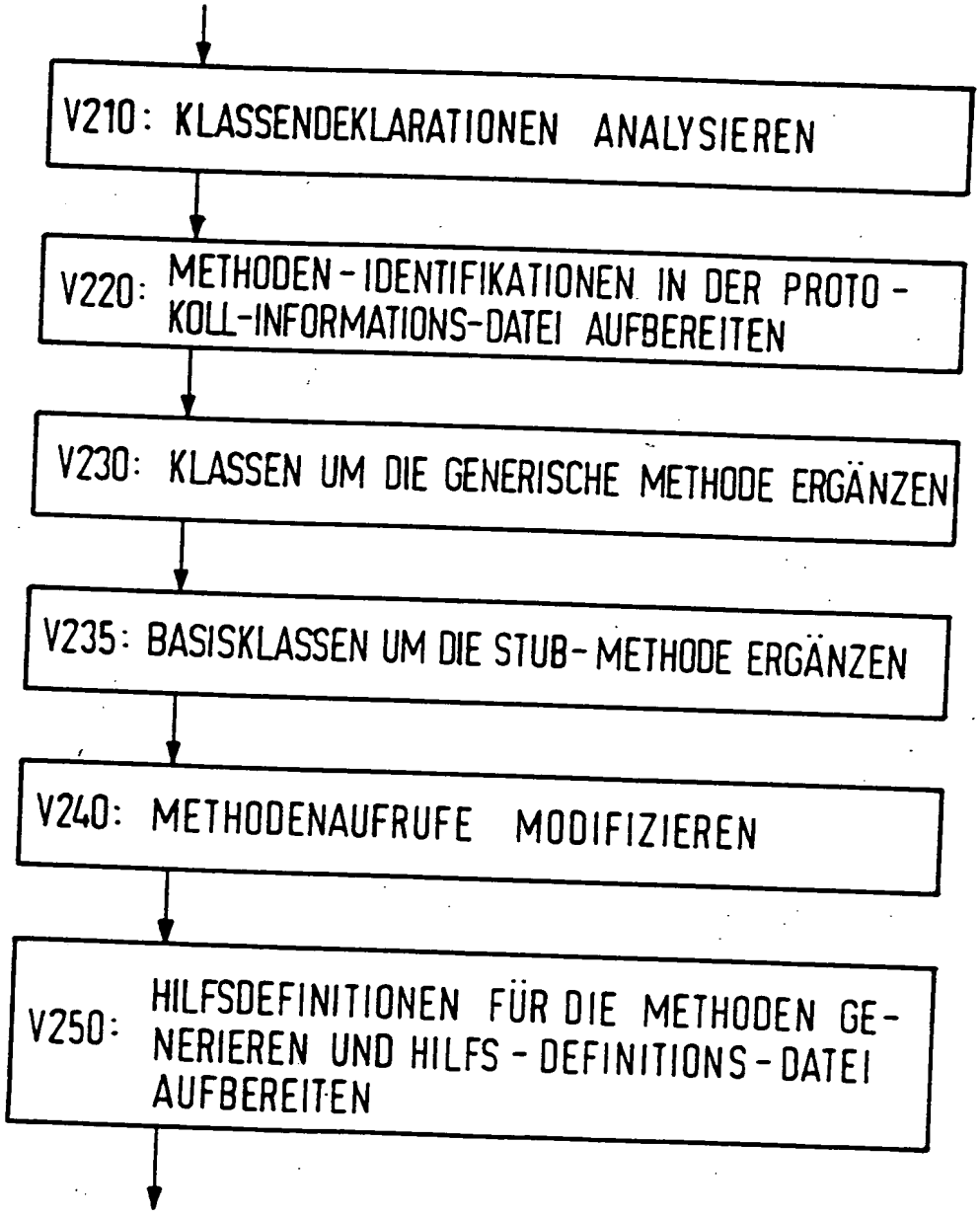
Nummer: DE 41 31 380 A1
Int. Cl.[5]: G 06 F 9/45
Offenlegungstag: 25. März 1993

FIG 2



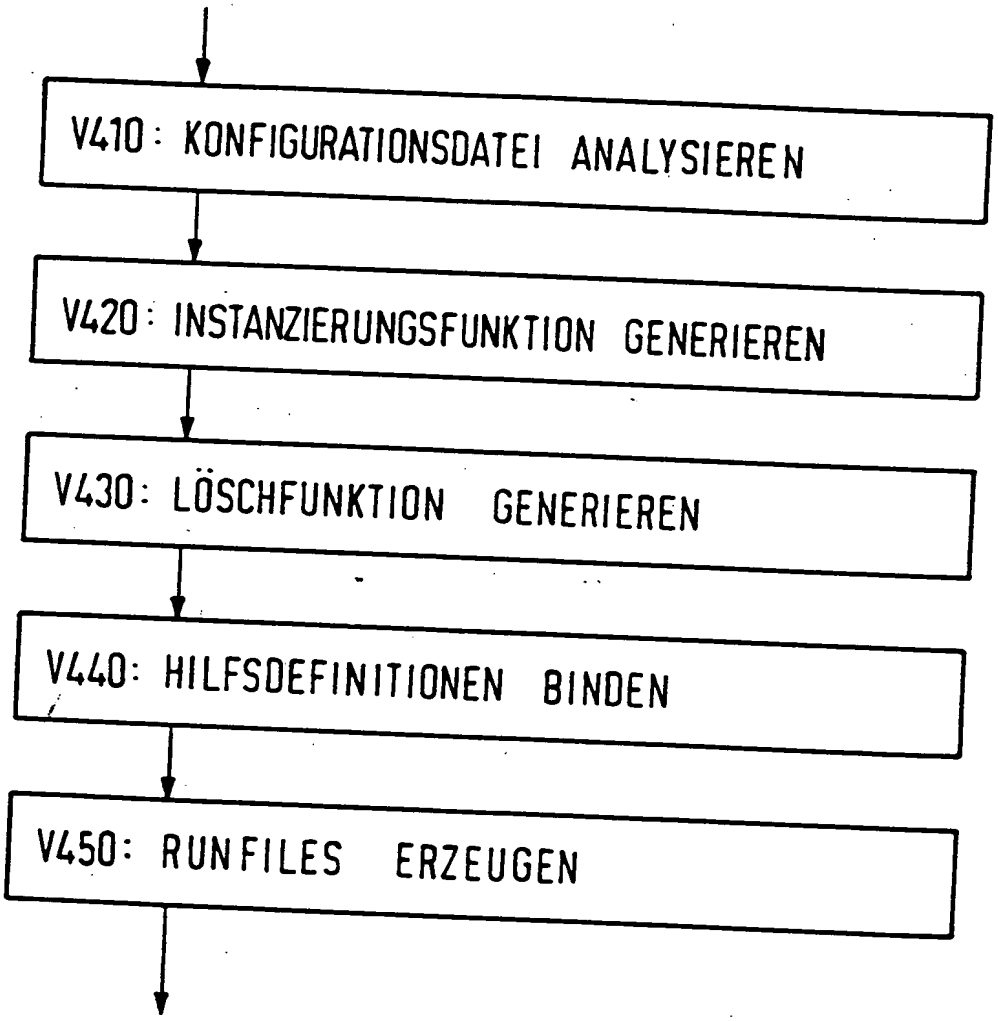
Nummer:
Int. Cl.[5]:
Offenlegungstag:
DE 41 31 [illegible]
G 06 F [illegible]
25. März 1993

FIG 3



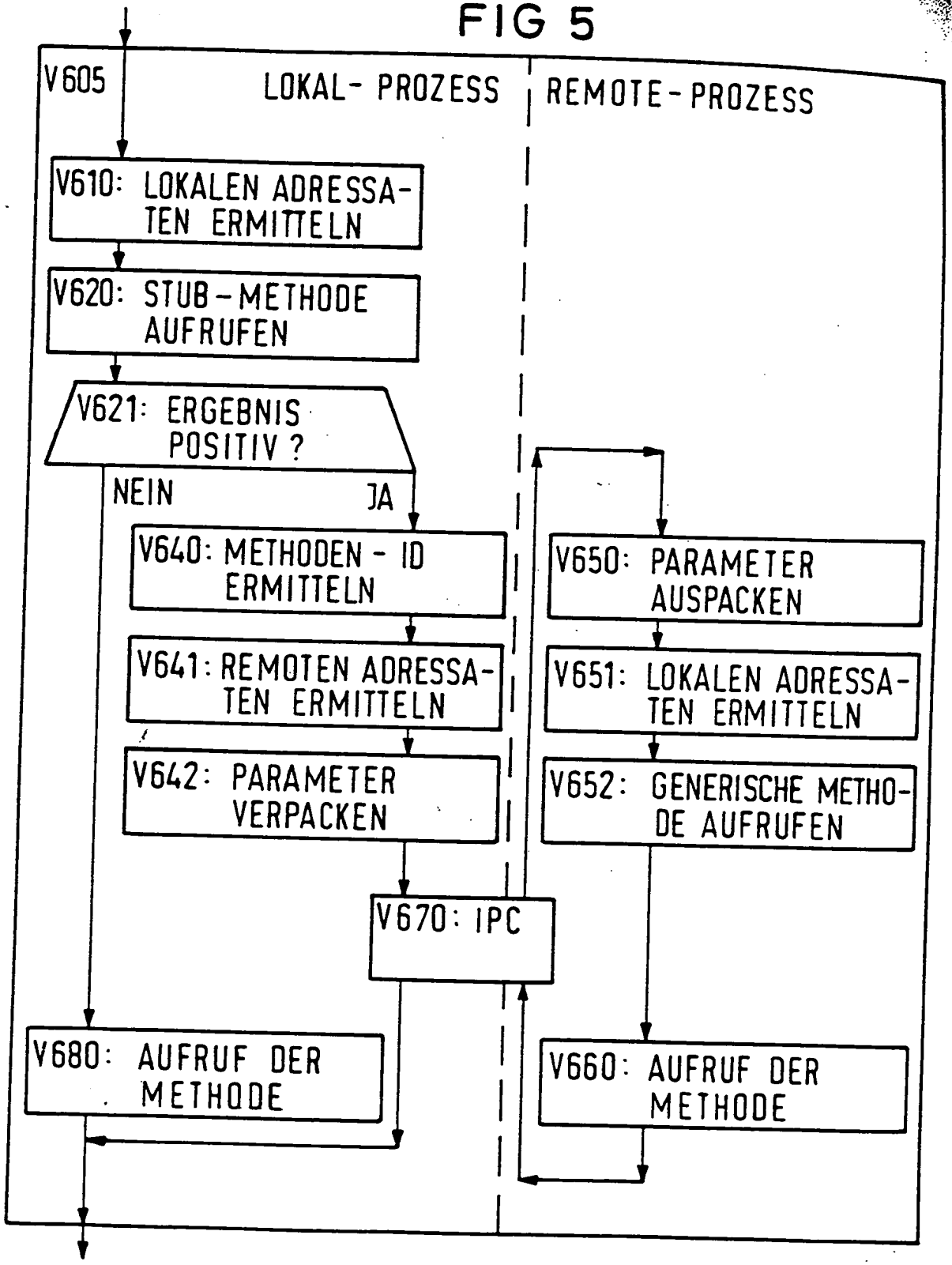
Nummer: DE 41 31 380 A1
Int. Cl.[5]: G 06 F 9/45
Offenlegungstag: 25. März 1993

FIG 4



Nummer:
Int. Cl.[6]:
Offenlegungstag:

FIG 5



Nummer: DE 41 31 380 A1
Int. Cl.[5]: G 06 F 9/45
Offenlegungstag: 25. März 1993

FIG 6

<Objektpointer> -> <Methodenname> (<Parameter>)

<Objektpointer>: Referenz auf das Objekt, an das die Nachricht gesendet wird
<Methodenname>: Bezeichnung der Nachricht
<Parameter >: Parameterteil für die Nachricht

FIG 7

((<Objektpointer> -> Vstub( ))? (SX_SEND ((_CSXobj*)
<Objektpointer>, < Methoden-ID>, <Parameterstring>, <Parameter>):
<Objektpointer>-><Methodenname>(<Parameter >))

<Methoden-ID>: Methoden-Identifikation zur systemweit eindeutigen Identifizierg. v. Methoden v. Klassen d. Applikation
<Parameterstring>: Zeichenstring zur Identifizierg. v. Parametern beim Einpacken sowie Auspacken v. Parametern

FIG 8

(SX_SEND ((_CSXobj*)<Objektpointer>, <Methoden-ID>,
<Parameterstring>, <Parameter>)

FIG 9

<Objektpointer> -> Vstub ( ))

Nummer: DE 41 31 380 A1
Int. Cl.[5]: G 06 F 9/45
Offenlegungstag: 25. März 1993

FIG 1